# A model-based design flow for CAN-based systems

Alexios Lekidis, Marius Bozga, Didier Mauuary, Saddek Bensalem

# A model-based design flow for CAN-based systems

Alexios Lekidis[1], Marius Bozga[1], Didier Mauuary[2], Saddek Bensalem[1]
[1]UJF-Grenoble 1 / CNRS-VERIMAG, [2]Cyberio

**Abstract. This paper introduces a novel approach for systematical development of CAN-based systems with guaranteed functional correctness and optimal performance. This approach relies on formal methods for faithful modeling and analysis of such systems, whilst taking into consideration the effects of critical parameters, such as bit stuffing and buffer utilization. As a proof of concept, the approach has been applied on existing benchmarks simulating realistic automotive networks. The results are similar to ones obtained using domain-specific tools e.g. NETCARBENCH. Moreover, this work creates new perspectives and reveals potential application for the generation of optimal device configurations for the recently developed CAN FD protocol.**

## 1. Introduction

Controller Area Network (CAN) [1] has emerged as a dominant network technology over the last 20 years, mainly due its robustness and cost-effectiveness. Nonetheless, the design of complex CAN-based systems remains challenging. To facilitate their design, high-level protocols built on top of CAN, such as CANopen [2] and DeviceNet [3], have been proposed. These protocols offer primitives to organize and to abstract the complexity of low-level communication in a CAN network. Complex networked systems can be much easily built but, however, such systems remain difficult to validate a priori. Previous studies have pointed out the importance of conformance testing [4] [5]. Nevertheless, testing occurs only late in the development cycle and requires the final system implementation. Earlier detection of functional errors as well as earlier performance analysis is paramount for successful design.

In this work, we propose a novel design flow for CAN-based systems using the BIP component framework [6]. This flow follows the general principles of rigorous design previously introduced in [6]. It has several key features, namely it is:

- *model-based*, that is, both the application software and the mixed networking hardware/software system descriptions are modeled by using a single, semantic framework. As stated in [6], this allows maintaining the coherency along with the flow by proving that various transformations used to move from one description to another preserve essential properties.
- *component-based*, that is, it provides primitives for building composite components as the composition of simpler components. Using components reduces development time by favoring component reuse and provides support for incremental analysis and design.
- *tool-supported*, that is, all steps in the design flow are realized automatically by tools. This ensures significant productivity gains, in particular due to elimination of potential errors that can occur in manual transformations.

The BIP design flow is unique as it uses a single semantic framework to support application and system modeling, validation of functional correctness, performance analysis on system models and code generation in distributed platforms. Building faithful system models is mandatory for validation and performance analysis of complex applications deployed on distributed networked platforms.

Generally, model-based design is getting increasing acceptance in many application domains nowadays. For example, [7] proposes a design flow for automotive systems. [8] presents a system-level design flow for building general automation and control systems. Nonetheless, to the best of our knowledge, no specific model-based design flows exist

for CAN-based systems. Furthermore, the existing ones such as [8] can be hardly adapted due to their inherent complexity and limitations for system-level modeling and analysis (e.g. long simulation time).

This paper is structured as follows. Section 2 presents briefly the underlying framework and details the key steps of the proposed design flow. Section 3 presents the model of the CAN protocol and the modeling principles of CAN-based systems in BIP. Section 4 provides experimental results on existing benchmarks and Section 5 discusses further extensions and perspectives for the work.

## 2. Design Flow for CAN-based systems

The design flow is based on the Behavior – Interaction - Priority (BIP)[1] framework and the associated tools for analysis and performance evaluation [6]. BIP provides a general component construction methodology facilitating the development of rigorous, trustworthy and correct-by-construction systems. Component construction in BIP is layered (Figure 1). The first layer (Behavior) consists of extended finite-state automata or Petri-Nets which models the basic processes, activities or functionalities (termed as atomic components) of the system. Every transition is labeled by an action name (termed as port), a guard and a function operating on the atomic component data. The ports are used in the second BIP layer (Interaction) which defines strong or loose synchronization, associated with data exchange, between atomic components. Interactions between components are defined by connectors. The third BIP layer (Priority) is used to restrict the non-determinism between simultaneously enabled interactions. A set of atomic (and composite) components can be composed by using successive application of interactions and priorities and encapsulated into a composite component.


*Figure 1: BIP layered component model*

[1] http://www.bip-components.com

The design flow for CAN-based systems is illustrated in Figure 2. It involves the following main steps:

I. **Translate the application software to BIP**. Applications running on top of CAN networks can be a priori developed using (possibly multiple) domain specific languages and/or particular programming models. The translation ensures their representation in a rigorous semantic framework, which is a prerequisite for any reliable and meaningful analysis.

II. **Modeling the CAN network in BIP**. The obtained BIP model represents the underlying network hardware and the network protocols running on it. This model is needed to capture the constraints imposed by the communication network on any potential application running on top. The model of the CAN network focuses on critical functional and timing aspects, including the CSMA/CA media access control, timing at the bit level, connectivity, scheduling policies for frames, correct frame identifier allocation etc.

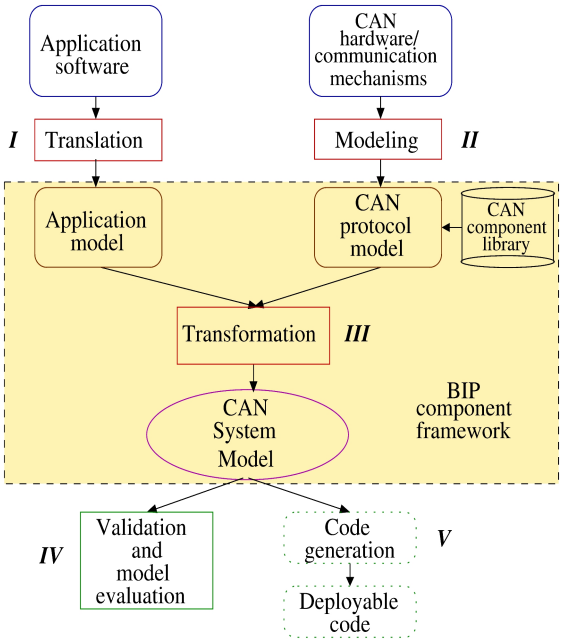


*Figure 2: Design flow for CAN-based systems*

III. **Construction of the system model in BIP**. The system model is intended to represent the entire mixed SW/HW system, that is, the application software running on top of the CAN network. This model is directly derived by a combined transformation of BIP models obtained in steps I and II using additional deployment

information, which is, the allocation and scheduling of various software modules onto network nodes. The key addition of this transformation is to meaningfully integrate the network (hardware) constraints on the application (software) model.

IV. **Functional correctness and performance analysis**. In the proposed flow, functional correctness means formal verification of safety properties (including deadlock-freedom) on the system model. Such properties can be proved using the D-Finder tool [9] and/or tested using the BIP simulation based tools. Performance analysis deals with verification of extra-functional QoS properties. These properties are proved using statistical-model checking [10].

V. **Code generation.** The BIP tools allow the generation of platform dependent C/C++ code from the CAN system model obtained in step III. This code can be used either for execution in the CAN network hardware components or for simulations with the dedicated simulation-based tools.

## 3. CAN-based system modeling

In this section, we focus on two key steps of the design flow, namely the modeling of the CAN networks (step II) and the construction of the system model (step III).

### 3.1 BIP Model of CAN Networks

Our model of the CAN network is representing the functionality of the classic CAN protocol [1]. Moreover, it is restricted to the Basic CAN [11], that is, a single transmit and a single receive buffer are used for the transmission and the reception of the frames accordingly. The model is also compliant with the High Speed physical layer standard [11], due its higher baud rate and interoperability with the higher-level protocols mentioned in section 1. Finally, the current version is not modeling transmission errors.

The BIP model of a CAN network is built using two types of components, namely the CAN station and the CAN bus components. CAN stations mediate the frame transmission on the CAN bus. They are later connected to application components. The CAN bus is modeling the arbitration and the broadcast mechanisms of every frame to all the connected CAN stations. The frame transmission process consists of two steps. First data are transmitted to the CAN bus and then broadcasted to all the stations, including the sender. Strong synchronization between all the CAN stations and the CAN bus is used for the sending of each frame field.

Each frame sent over the CAN bus can be of two types: data transmission (data frame) or data request (remote frame). In both cases it is represented by the tuple {*arb*, *rtr*, *ide*, *length*, *payload*}, whose meaning is as follows:

- *arb* is the frame identifier
- *rtr* is the Remote Transfer Request (RTR) bit
- *ide* is the IDentifier Extension (IDE) bit
- *length* contains the length of the data to be sent
- *payload* contains the data

**BIP Model of CAN stations.** CAN stations are composite components consisting of two atomic components: the CAN Controller and the CAN Filter. These components are responsible for the frame transmission to the CAN bus (*REQUEST* interaction) and the frame transmission to the application (*RECV* interaction) accordingly. The Controller component uses a transmission queue, in order to store the pending frames, that is, received from the application and waiting to be sent over the Bus. The queuing policy can either be of type First-in-First-Out (FIFO) or High Priority First (HPF), where frames are selected according to their priority. The selection is application-specific.

The Controller component (Figure 3) is modeled as a Petri-Net, which (1) receives frames from the application and (2) sends or receives frames from the CAN bus component. The transmission process is initiated through the *REQUEST* port, which stores the received frame in the transmission queue. If the Controller has a frame to send, the transmission cycle begins (*SOF* port). Next, in the arbitration phase, labeled by the *ARBITRATION* port in the model, every Controller sends its identifier (*arb* field) to the CAN bus. The minimum identifier “wins” the arbitration and gets broadcasted

to all of them[2]. The Controller with the minimum identifier is allowed to proceed with the transmission of the *length* and *payload* fields, while all the others are receiving them. The end of the transmission cycle is denoted by the EOF port. Throughout this cycle's duration the *REQUEST* port is always available, ensuring the seamless frame reception from the application. If at least one receiving CAN station receives the frame fields correctly, the sending Controller will stop retransmitting its frame. The receiving Controllers send the frames to the Filter component through the port *RECV*.

The acceptance filters receive every frame from the Bus, in order to either deliver it to the application or ignore it. Thus, the Filter component is receiving all the frames through an interaction involving its *HANDLE* port and the *RECV* port of the Controller component, such that only the needed frames are delivered to the application. It checks accordingly their identifier to a list of identifiers provided by the application. If the identifier belongs to the list, the frame is transmitted through the transition *RECV*, otherwise it is discarded.

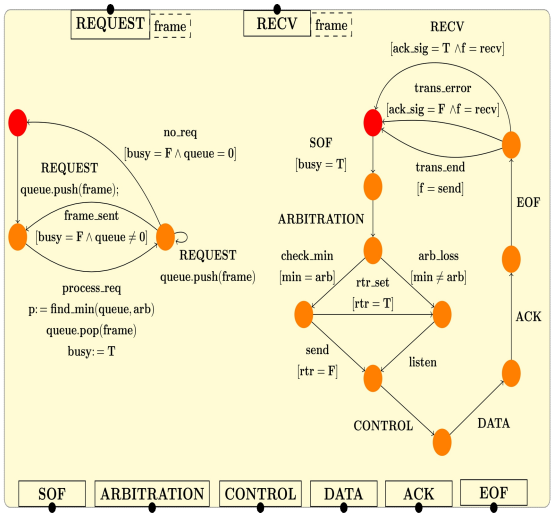


*Figure 3: CAN Controller component*

**BIP Model of the CAN bus.** The CAN bus component is using two groups of ports for its interactions:

- The TICK port denoting one time step advance and,
- The SOF, ARBITRATION, CONTROL, DATA, ACK, EOF ports used for interaction with the Controller component

As shown in Figure 4 the CAN bus is receiving the frame fields {*arb*, *rtr*, *ide*, *length*, *payload*} from the Controller component. A significant difference to the Controller is the modeling of the time step needed for the transmission of each frame field, denoted by the port *TICK* in the model. One tick corresponds to the time needed for the transmission of one bit ($2_{bit}$). The role of the CAN bus is to synchronize all the CAN stations. During the transmission cycle it interacts with all the CAN station components through the *SOF* port. The identity of the data frame sent to the Bus is determined through a check on the *ide* field, providing information about the number of bits transmitted through the *ARBITRATION* port. The resulting value is 12 for a standard frame and 32 for an extended representing the time needed for the arbitration phase, accordingly stored in variable *g*. The time duration for the transmission of the payload field (*DATA* port) will depend on the value of the length field received through the *CONTROL* port (6-tick time duration). The checksum computation results in a 16-tick time duration. The transmission cycle ends through the *EOF* port, which along with the ACK port correspond to a 9-tick time duration. The presence of the Interframe space (IFS) between consecutive frame transmissions is used to avoid Bus overload occurrences and corresponds to a 3-tick time duration in the model. After this time elapses the control returns to its initial state (*ifs* port). The overall frame transmission time in the model is given by:

$$C_{frame} = (32 + g + 8 \times length)_{2\ bit} \quad (1)$$

However bit-stuffing protocol violation errors [1] may increase the aforementioned time by:

$$C_{stuffing} = \left\lfloor (23 + w + 8 \times length - 1)\frac{s}{100} \right\rfloor_{2\ bit} \quad (2)$$

where $w = g - 1$, since the remote request bit is not subject to stuffing, $_{2\ bit} = 1$ and $s \in [1, 25]$ is a parameter of the model, denoting the number of stuffed bits for every frame. The value of *s* for the

[2] If a Controller has no frame to send its identifier will be automatically set to $2^{11}$ for the standard frame and $2^{29}$ for the extended

transmission of a frame is handled by a given distribution on its interval. Related to the analysis provided in [12], our model is not considering the IFS field as part of the frame and the worst-case transmission time is provided with *s* equal to 25.

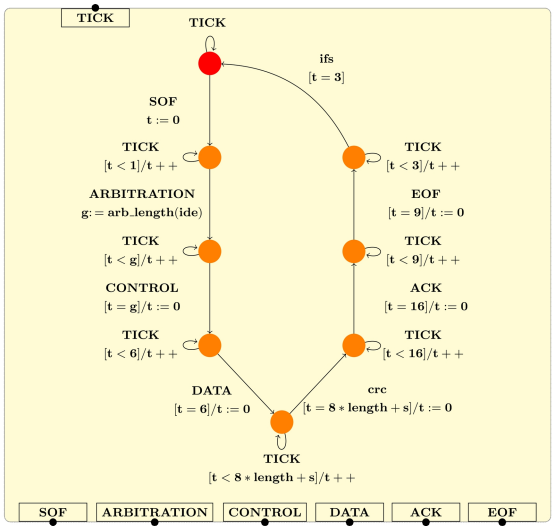


*Figure 4: CAN bus component*

### 3.2 CAN-based System Model

Figure 5 illustrates the system model architecture. The application software consists of a number of Device components (upper layer). They are interacting only with CAN stations for the transmission or reception of frames.

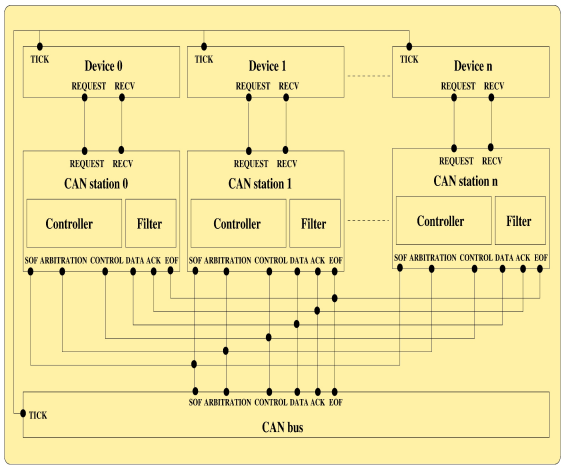


*Figure 5: System model in BIP*

Device components are application specific and their functionality is directly derived from the application software. They are completely decoupled of the CAN protocol model, described in the previous section, allowing the separation of concerns. Device components are modeled as Petri-Nets, separating their interactions with the CAN station. Concrete examples of such components for benchmark applications are presented in the following section.

Communication over the Bus requires their composition, in order to form the system model (step III of section 2). To achieve this we apply interactions between the Device and the CAN station components. These interactions involve frame transmission through the *REQUEST* port and frame reception through the RECV port.

## 4. Validation and experiments

The conducted experiments are focusing on validation and performance evaluation for two case studies. The first concerns a deterministic powertrain network benchmark [13], triggering periodic data transmission through the CAN bus. For this case study we compare our approach with existing domain-specific tools, such as NETCARBENCH [14]. The comparison is done in terms of accuracy and simulation time. The second case study is an extension of the first one, where frame transmission is subject to probabilistic distributions applied on frame periods and bit-stuffing. This model exceeds the simulation capabilities of Netcarbench. It can also be analyzed using the recently incorporated statistical-model checking tool of the BIP toolset [10]. Applications are represented as a collection of Device components (Figure 6). These components are atomic and contain a transmission and a reception part. Figure 6 illustrates the former part. Frame transmission is handled by the *REQUEST* port, whereas frame reception by the *RECV* port. Each frame is triggered when some specific period is reached (port *generate*). This is achieved by consecutively incrementing variable *t* through the port *TICK*, until it is equal to the minimum period of the array *P*, which stores the periods for all the frames. The size of *P* is a model parameter, denoted as *N*. The periods may be fixed (Figure 6a), or differ according to a transmission margin, chosen from a probabilistic distribution and stored in the array *m* (Figure 6b). The resulting period is stored in the array *D*. The minimum period in both cases is first calculated in the initial state and afterwards iteratively.

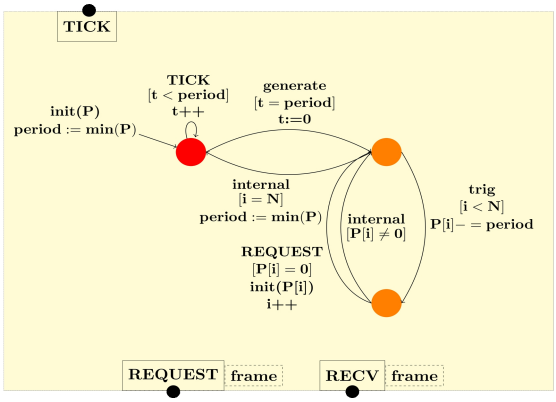


*Figure 6a: Periodic Device component*

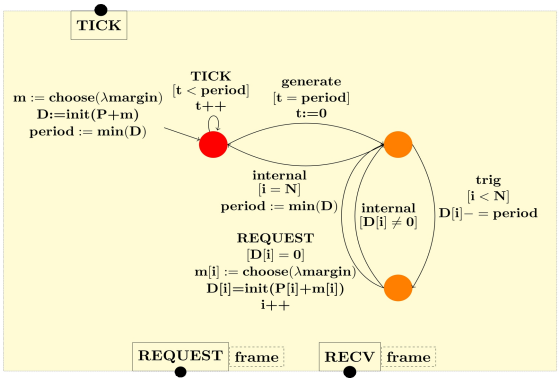


*Figure 6b: Stochastic Device component*

**Case study 1:** The deterministic powertrain network benchmark was generated by NETCARBENCH. It consisted of 5 Electronic Control Units (ECUs) communicating over a Bus with a bit-rate of 500kbit/s. The queuing policy used was HPF and the observed Bus load was 13.8%, distributed approximately equal in every ECU. Bit-stuffing was fixed to 10%, meaning s was equal to 10 for every frame in equation (2). Transmission offsets and clock drifts were not considered in this example. All parameters concerning the frame identifier, period and payload are provided in Table 1. Our analysis focused on the frame response times using two methods. The first method applied the BIP design flow on the generated benchmark, to construct the BIP system model and then to analyze it. The derived translation represented the entire SW/HW system, reflected by the benchmark. The obtained system model was accordingly simulated using the associated simulation-based tools. The second method provided the generated benchmark as input to RTaW-Sim [15], a discrete-event fine-grained CAN bus simulator.

The system model in BIP contained 15 atomic components for the CAN protocol model and 5 atomic components for the application model. It also used 60 connectors (40 for the CAN protocol and 20 for the application model). The total number of transitions in the system was 255 (210 for the CAN protocol and 45 for the application model). Overall the model totals about 1250 lines of BIP textual code.

| ECU | CAN ID | Period (ms) | Payload |
|---|---|---|---|
| 1 | 189 | 10 | 5 |
| | 200 | 20 | 1 |
| | 269 | 50 | 2 |
| | 298 | 50 | 8 |
| | 533 | 100 | 6 |
| | 685 | 2000 | 8 |
| 2 | 328 | 20 | 6 |
| | 371 | 100 | 8 |
| | 379 | 20 | 8 |
| | 477 | 50 | 5 |
| | 506 | 200 | 8 |
| 3 | 262 | 20 | 7 |
| | 427 | 50 | 7 |
| | 472 | 100 | 6 |
| | 492 | 100 | 7 |
| | 774 | 2000 | 8 |
| | 977 | 1000 | 8 |
| 4 | 159 | 20 | 6 |
| | 208 | 20 | 7 |
| | 321 | 50 | 7 |
| | 480 | 50 | 8 |
| | 502 | 100 | 4 |
| | 628 | 200 | 7 |
| | 690 | 2000 | 8 |
| | 776 | 1000 | 8 |
| 5 | 260 | 20 | 4 |
| | 307 | 50 | 6 |
| | 370 | 100 | 5 |
| | 473 | 50 | 6 |
| | 724 | 200 | 7 |

*Table 1: Network configuration parameters*

Figure 7 illustrates the results obtained using the two methods, where the analysis was focused in three categories, that is, minimum, average and worst-case frame response times. The results were identical for both methods, in all the aforementioned categories. From the conducted analysis we can also note that approximately 55% of the frames had a deterministic response time, where the remaining 45% had a fixed queue waiting time, due to higher priority frame transmission.

A real system time of 1 hour was simulated in 5 minutes and 30 seconds using the BIP simulator and in 13.5

seconds using the RTaW-Sim simulator. The observed divergence occurred due to the difference in the simulation models. The BIP simulator is state-based, whereas RTaW-Sim is an event-based simulator. Nevertheless, we are currently introducing existing model transformations [16] in the BIP system model, in order to improve the simulation time.

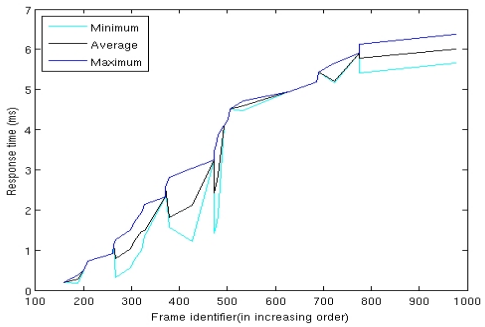


*Figure 7: BIP/RTaW-Sim frame response times for the powertrain network*

**Case study 2:** The second case study introduced a stochastic behavior to the previous benchmark. First, we added a probabilistic margin for every period. Each margin followed a Poisson distribution based on a mean rate equal to 1/10 of each period. Moreover, parameter *s* in equation (2) was not fixed, but varied according to a uniform distribution in the range [1,25]. Consequently, each frame transmission had a different bit-stuffing error. The results, shown in Figure 8, are also divided in the three aforementioned categories. As it is observed, in average all the frames have a very small waiting time. However, due to the non-deterministic behavior of the system, response times cannot be described only though the previous analysis. Therefore, in Figure 9 we focus on a particular frame, in order to show the probabilistic variation of the obtained response times.

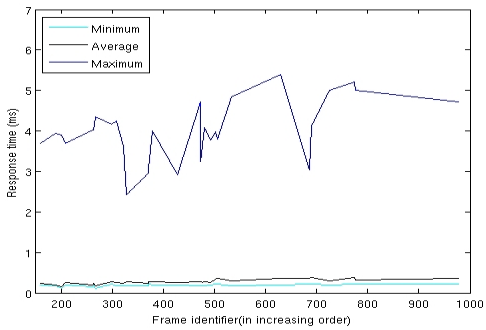


*Figure 8: BIP frame response times for the stochastic powertrain network*

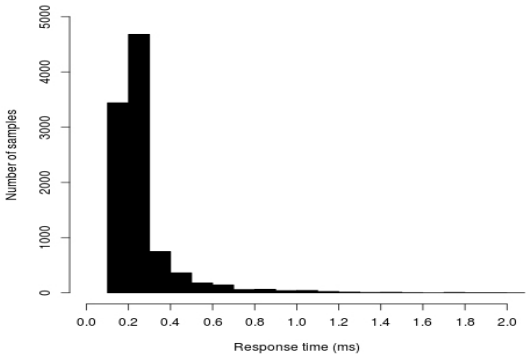


*Figure 9: Response time distribution of a frame in the stochastic powertrain network*

## 5. Conclusion and ongoing work

In this paper we presented a rigorous design flow for the correct construction of CAN-based systems. We explained the main principles of the automatic translation from the application software and the CAN communication mechanisms along with the underlying hardware to a BIP system model. This model allows the separation of hardware and software design issues. Furthermore, it can be used for performance analysis and for generation of platform dependent C/C++ code.

For the time being, we are investigating further extensions to the CAN network model, in order to provide the full functionality of the recently developed CAN FD protocol [17]. To accomplish that we will add the *edl* and the *brs* fields to the frame mentioned in section 3. The former denotes the Extended Data Length bit of the CAN FD frame, whereas the latter indicates if the bit rate is switched from the standard to the alternate bit rate during the transmission of the *payload* field. In this case, the time needed for the transmission of one bit ($^2{}_{bit}$) will be shorter than 1 tick and handled by the parameter *tswitch* in the model. This parameter denotes the switch factor between the alternate and the standard bit-time and its value depends on the selection of the CAN network hardware components. Finally, considering the bit stuffing analysis of [17] equation (2) will differ such that:

$$C_{stuffing} = \left( \left\lfloor (7 + w + 8 \times length) \frac{s}{100} \right\rfloor + 1 + \left\lfloor \frac{15}{4} \right\rfloor \right) \tau_{bit} \Leftrightarrow$$

$$C_{stuffing} = \left( \left\lfloor \frac{7 + w + 8 \times length}{s} \right\rfloor + 4 \right)^2{}_{bit} \quad (3)$$

Since the aforementioned extensions are only related to the CAN protocol model, the application software model will remain unaffected. In the scope of these extensions, we plan to develop a similar design flow for CAN FD systems. The system model will be accordingly tested using the BIP simulation tools, in order to obtain optimal configuration parameters for every device of the application software. Consequently, these parameters will be used to generate device configuration files along with the platform dependent C/C++ code.

Alexios Lekidis
UJF/Verimag
2 Avenue de Vignate.
38610 Gières – France
Alexios.Lekidis@imag.fr
http://www-verimag.imag.fr/~lekidis/

Marius Bozga
UJF/Verimag
2 Avenue de Vignate.
38610 Gières – France
Marius.Bozga@imag.fr
http://www-verimag.imag.fr/~bozga/

Didier Mauuary
Cyberio
6 bis Chemin des prés Inovallée
38240 Meylan – France
Didier.Mauuary@cyberio-dsi.com

Saddek Bensalem
UJF/Verimag
2 Avenue de Vignate.
38610 Gières – France
Saddek.Bensalem@imag.fr
http://www-verimag.imag.fr/~bensalem/

## References

[1] Robert Bosch. CAN specification version 2.0. Robert Bosch GmbH, Postfach, 300240, 1991.

[2] CANopen CiA. Application layer and communication profile, Draft Standard 301, 2002.

[3] Specification, DeviceNet. "Release 2.0, including Errata 4." April 1 (2001): 1995-2001.

[4] Zeltwanger Holger, "CAN Implementations and Conformance Testing" , SAE Technical Paper 1999-01-1273, 1999

[5] Road Vehicles- Controller Area Network (CAN)- Conformance Test Plan, ISO 16845:2000

[6] Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based design using the BIP framework. IEEE Software, Special Edition – Software Components beyond Programming – from Routines to Services 28(3), 41–48 (2011)

[7] Sangiovanni-Vincentelli, Alberto, and Marco Di Natale. "Embedded system design for automotive applications." Computer 40.10 (2007): 42-51.

[8] Yang, Y., Pinto, A., Sangiovanni-Vincentelli, A., & Zhu, Q. (2010, November). A design flow for building automation and control systems. In Real-Time Systems Symposium (RTSS), 2010 IEEE 31st (pp. 105-116). IEEE.

[9] Bensalem, S., Bozga, M., Nguyen, T. H., & Sifakis, J. (2009, January). D-finder: A tool for compositional deadlock detection and verification. In Computer Aided Verification (pp. 614-619). Springer Berlin Heidelberg.

[10] Saddek Bensalem, Marius Bozga, Benoit Delahaye, Cyrille Jegourel, Axel Legay, and Ayoub Nouri. Statistical model checking QOS properties of systems with SBIP. In Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, Pages 327–341. Springer, 2012

[11] Pfeiffer, Olaf, Andrew Ayre, and Christian Keydel. Embedded networking with CAN and CANopen. Copperhill Media, 2008.

[12] Davis, R. I., Burns, A., Bril, R. J., & Lukkien, J. J. (2007). Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Systems, 35(3), 239-272.

[13] Cook, J. A., Sun, J., Buckland, J. H., Kolmanovsky, I. V., Peng, H., & Grizzle, J. W. (2006). Automotive powertrain control—A survey. Asian Journal of Control, 8(3), 237-260.

[14] Nicolas Navet Christelle Braun, Lionel Havet. NETCARBENCH: A BENCHMARK FOR TECHNIQUES AND TOOLS USED IN THE DESIGN OF AUTOMOTIVE COMMUNICATION SYSTEMS. 7th IFAC International Conference on Fieldbuses and Networks in Industrial and Embedded Systems, 7(45):321–328, 2007.

[15] Nicolas Navet, Aurelien Monot, Jörn Migge, et al. Frame latency evaluation: when simulation and analysis alone are not enough. In 8th IEEE International Workshop on Factory Communication Systems (WFCS2010), Industry Day, 2010.

[16] Bozga, Marius, Mohamad Jaber, and Joseph Sifakis. "Source-to-source architecture transformation for performance optimization in BIP." Industrial Informatics, IEEE Transactions on 6.4 (2010): 708-718.

[17] Robert Bosch GmbH, CAN with Flexible Data-Rate, Specification, Version 1.0, April 2012,http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can_fd_spec.pdf